

# PadhAI: Sequence Learning Problems

## One Fourth Labs

### Setting the context

Previously, we learnt about two types of DNN models.

- 1) Fully Connected Neural Networks
- 2) Convolutional Neural Networks

In both of these networks, the output of a given input didn't depend on the previous inputs given to the model. In a nutshell **Outputs are completely independent of the previous inputs.**

**Also, all the inputs are of the same size.**

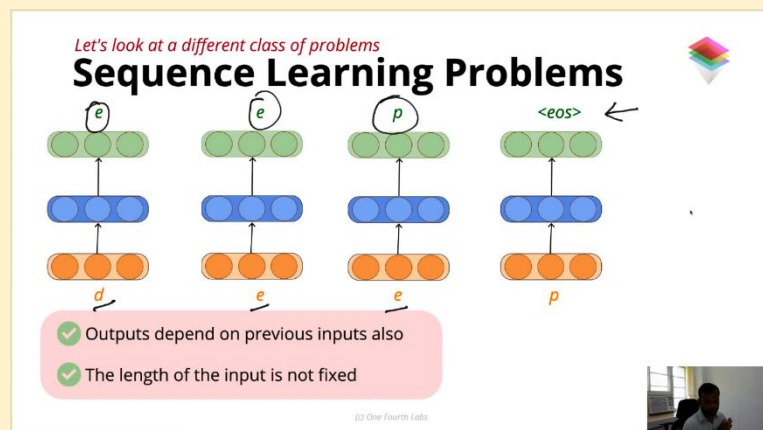
If all the inputs are not of the same size, we rescale all the input data to a particular size. For example, we rescale all the input images of various sizes to 30×30.

The above mentioned two properties of FNN and CNN don't hold for a class of problems, in such problems the **current output depends upon the previous inputs, and also the size of each input is not the same.** In this module we are going to discuss about such class of problems and models (networks) or solutions which can handle such problems.

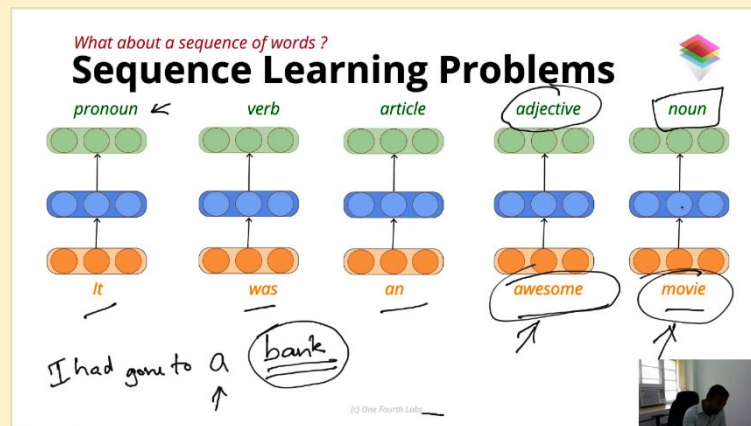
### Introduction to Sequence learning problems

Sequence learning problems are problems in which the **current output depends upon the previous inputs and the length(size) of the inputs is not necessarily fixed.**

#### a. Sequence Learning problems using text data.

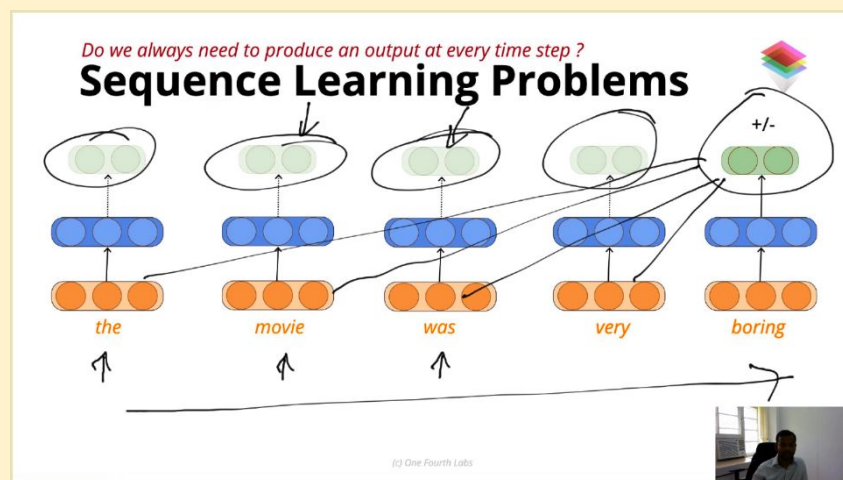


1. In the above fig. we can see that each predicted letter (green output layer) depends upon the previous inputs (orange input layer). <eos> is end of sequence. i.e., the letters of the word which was to be predicted are all typed and the sequence of predicted letters ends.



- The above fig. represents a problem which is known as **Part of Speech tagging problem**, it is a sequence learning problem. In this problem, for a given sequence of words we need to predict the part of speech tag for each word in the sequence. Let's say for example, in the above fig. the word **awesome** is an **adjective**, which makes us predict that the next word in the sequence, whatever it might be, is going to be a **noun**. So, the prediction of the word **movie** as a **noun**, **depended** upon the **previous input** word **awesome**.

In the sentence “**I had gone to a bank**”, the word **bank** can be both **noun** or a **verb**. Looking alone at the current given input sometimes makes it difficult or nearly impossible to predict the output, so we need to depend upon the previous inputs to predict the output for the given input. In our example, the input word before the current input word ‘**bank**’ was ‘**a**’ which is an **article**, which lets us predict that the current input word **bank** is a **noun** and **not a verb**.

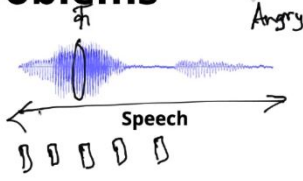


- The above problem is known as **Sentiment Classification** problem, in which we take the whole input sequence, a whole sentence, and produce an output (+ve or -ve sentiment) which depends upon all the inputs. We basically **ignore the outputs** which are produced at each time step (output produced by each input) and only consider the final output. **Individual inputs** or **few inputs cannot predict** whether the sentence is **negative or positive**, that's why we consider the final output, which **depends on every input** of the sequence. We assign a class (+ve or -ve) to a whole sequence of inputs.

## b. Sequence Learning problems using video and speech data.

What are some other types of sequences that we encounter ?

### Sequence Learning Problems

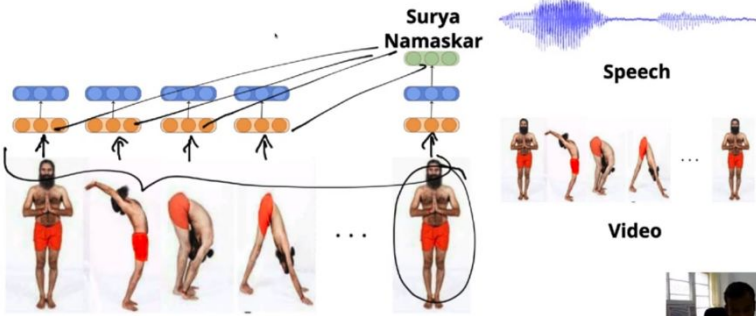


(c) One Fourth Labs

1. In **speech-based sequence learning problems** we take a whole audio clip and try to predict things like, whether the person is speaking in an angry or a happy or in any other tone, Which language he's speaking or what words or letters are there in the clip and many other such audio-based things are predicted. This requires a **sequence of inputs** to predict them accurately.

What are some other types of sequences that we encounter ?

### Sequence Learning Problems



(c) One Fourth Labs

2. In **video-based sequence learning problems** the **final output depends upon the previous input frames**. In the above shown fig. we cannot simply tell whether it is a Surya Namaskar or not only by looking at the final input frame or other intermediate input frames. The final output depends on the whole sequence of input frames and not on an individual or a bunch of frames. The number of frames may vary for different video inputs, because the time taken by different people to do Surya Namaskar might vary.

Above are the 3 types (Text, Speech and Video) of sequence learning problems in which deep learning is used to predict the labels at every time step or at the last time step.

## A wish-list for modelling sequence learning problems

Case 1: When we want an output at every time step.

1. **Same function** should be executed **at every time step**.
2. Function should be able to work with a **variable number of inputs**.
3. The function should consider **current as well as previous inputs**.

Case 2: When we want the output only at the end/ last time step.

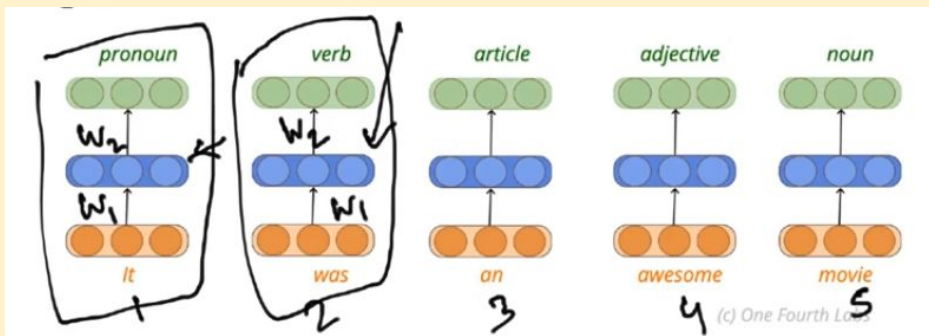
1. Function should be able to work with **variable number of inputs**.
2. Function should consider **current as well as previous inputs**.

Take away: We need to find such a function which satisfies the conditions for both the cases. This leads us to Recurrent Neural Networks.

## Intuitions behind RNNs

In our wish-list for modelling sequence learning problems we have 3 things.

**Same function** should be executed **at every time step**.



Orange Layer – Input Layer

Blue Layer – Hidden Layer

Green Layer – Output Layer

$$\begin{aligned}h_i &= \sigma(W_1 x_i + b_1) \\ y_i &= O(W_2 h_i + b_2) \\ i &= \text{timestep}\end{aligned}$$

$W_1$  and  $W_2$  are the weight matrices of layer 1 and layer 2 respectively.

$b_1$  and  $b_2$  are the bias vectors of layer 1 and layer 2 respectively.

$h_i$  is the output of hidden layer, applying a **non-linear function** at time step  $i$ .

$y_i$  is the final output obtained by the output layer by applying **softmax** to  $h_i$ .

## Change in notations in case of RNNs

$$s_i = \sigma(Ux_i + b)$$

$$y_i = O(Vs_i + c)$$

$i$  = time step

$s_i$  corresponds to  $h_i$  and is called **state**.

$U$  and  $V$  corresponds to  $W_1$  and  $W_2$  respectively

$b$  and  $c$  corresponds to  $b_1$  and  $b_2$  respectively.

**Parameter sharing:** This means that whatever function we execute at every time step should have the same parameters.

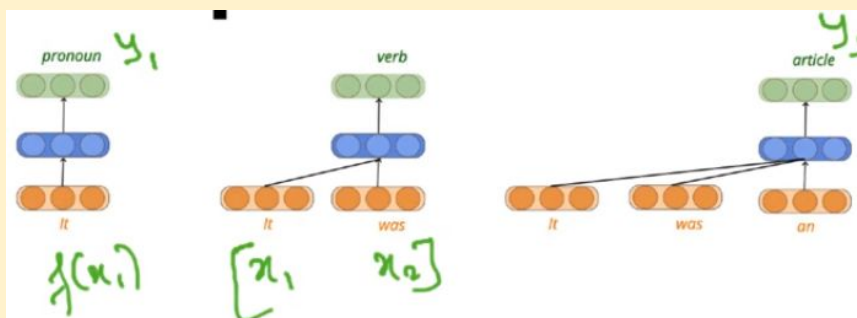
$$y_i = f(x_i, U, V, b, c)$$

Here  $x_i$  is the input.

The parameters  $U, V, b, c$  are the shared parameters.

Parameter sharing ensures that the function being executed at every time step is same, this satisfies our first point in the wish list.

1. Function should be able to work with a **variable number of inputs**.



$$y_1 = f(x_1)$$

$$y_2 = f(x_1, x_2)$$

$$y_3 = f(x_1, x_2, x_3)$$

$$y_n = f(x_1, x_2, x_3, \dots, x_n)$$

$n$  is time step.

So as to satisfy this point our **function should be able to work with a variable number of inputs**, to work with a variable number of inputs **the parameters of the function change at different time steps**. As a consequence of that we are **not able to share parameters**. This **violates the first condition of our wish list**.

**This does not serve our purpose(wish-list conditions) because**

1. The number of parameters at every time step increases.
2. Function being executed at every time step is different.
3. We are not able to deal with arbitrary input size.

**To satisfy the wish-list for modelling sequence learning problems we need to look for a different approach.**

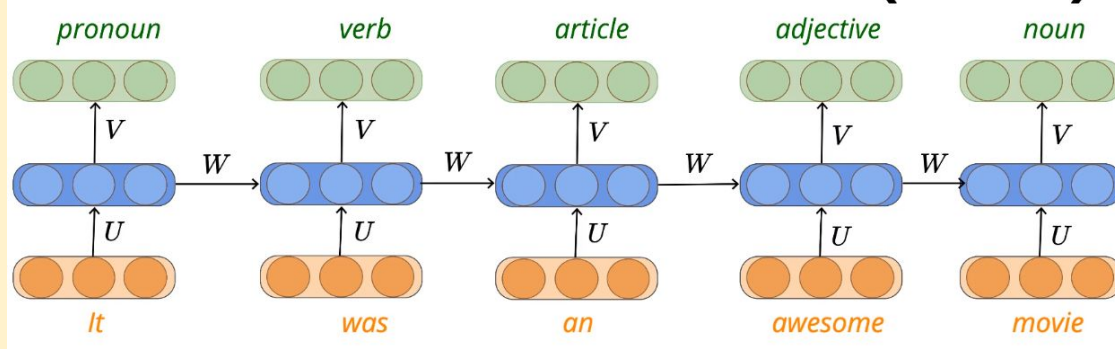
## The final solution

### Introducing RNNs

Our previous approach was only able to satisfy the condition that the **function should consider current as well as previous inputs** and failed to satisfy the other two conditions.

- ✓ Ensure that  $y_t$  is dependent on previous inputs also
- ✗ Ensure that the function can deal with variable number of inputs
- ✗ Ensure that the function executed at each time step is the same

## Recurrent Neural Networks (RNNs)





$$s_i = \sigma(Ux_i + Ws_{i-1} + b)$$

$$y_i = O(Vs_i + c)$$

RNNs satisfy all the conditions of our wish list.

**1. Same function should be executed at every time step.**

From the diagram above, we can see that at every time step the parameters **W**, **U**, **V**, **b** and **c** are the same. This means, at every time step the same function is being executed.

$$y_i = \hat{f}(x_i, s_{i-1}, W, U, V, b, c)$$

$x_i$  and  $s_{i-1}$  are the inputs and they can vary at every time step.

**2. The function should consider current as well as previous inputs.**

In the equation,

$$s_i = \sigma(Ux_i + Ws_{i-1} + b)$$

To ensure that the function take previous inputs also in consideration, we have included the term  $Ws_{i-1}$  in the equation of current state( $s_i$ ).

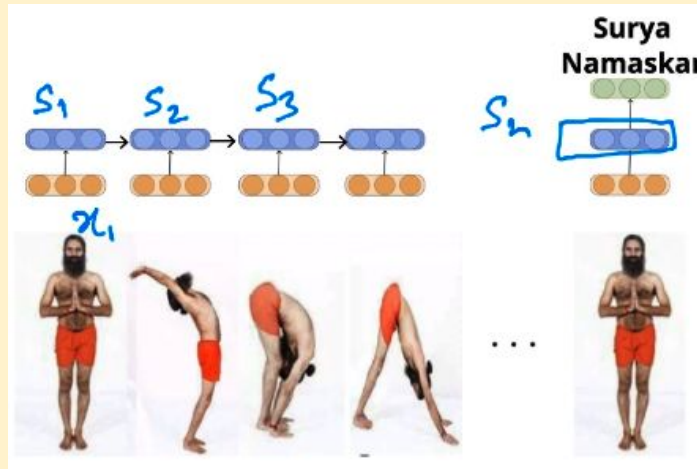
$s_{i-1}$  is the value of the previous hidden state, which depends on the input of previous time step, this enables our function  $y_i$ , it depends on the value of  $s_i$ , to consider previous inputs along with the current input.

$$y_i = O(Vs_i + c)$$

**3. Function should be able to work with a variable number of inputs.**

This is the by-product of both the previous conditions, if function remains the same at every time step and it considers current as well as previous inputs then the function would be able to work with any number of inputs.

For the second case of wish-list in which we need to calculate output at only last time step.



$y_i$  will not be computed at every time step, we just need to compute  $s_i$  at every time step until we reach the last time step, for that we just need the inputs  $x_i$  and  $s_{i-1}$  which we have already computed. After reaching the final/last time step, there we calculate the final output  $y_i$  with the formula

$$y_i = O(Vs_i + c)$$